Experiences Teaching an FPGA-based Embedded Systems Class

Stephen A. Edwards
Department of Computer Science
Columbia University
New York, NY 10027
Email: sedwards@cs.columbia.edu

***Abstract*— I describe a two-year-old embedded systems design course I teach at Columbia University. In it, the students learn low-level C programming and VHDL coding to design and implement a project of their own choosing. The students implement their projects using Xilinx FPGAs and tools running on Linux workstations.**

The main challenges the students face are understanding and complying with complex and often poorly-documented interfaces and protocols, personal time management, and teamwork. While all real-world challenges, this class is often the first time the students encounter them, which makes the class quite challenging, but very practical.

In this paper, I describe the structure of the class, the configuration of our teaching laboratory, some of the more successful projects, and give suggestions to instructors wishing to implement the class elsewhere.

I. Introduction

Embedded system design is a challenging problem that represents the future of digital system design. Moore's law and the relentless downward-spiraling cost of integrated circuits has made it possible to price very powerful computing artifacts at consumer levels, as the ubiquity of devices such as DVD players, digital cameras, and cell phones attest.

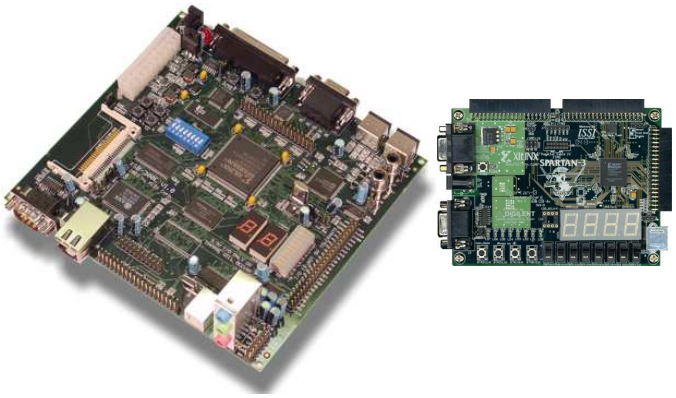
We in academia must follow this trend. As recently as fifteen years ago, wire-wrapping TTL parts was a sufficient introduction to state-of-the-art system design techniques; today's systems are orders of magnitude more complex. Teaching students to handle this complexity is the central challenge.

In this paper, I describe an FPGA-based embedded systems course I developed and teach at Columbia University. Following the suggestion of Frank Vahid, whose book [1] I used for the first year, I created this course to replace a microprocessor system design course that until 2003, had students assemble systems using Z80-based trainers with breadboards[1].

I wanted the students to learn hardware/software codesign, specifically the design of microprocessor systems and their peripherals. Students taking the course have experience with Java and C, know basic digital design, and have taken computer organization, which should have familiarized them with assembly language, but few have any experience integrating these skills.

Edwards is supported by an NSF CAREER award, an award from the SRC, and by New York State's NYSTAR program.

[1]To emphasize their antiquity, these trainers were labeled "Copyright 1985."



(a) (b)

Fig. 1. The two boards used in different incarnations of the class. (a) The XESS XSB–300E board, centered around a Xilinx Spartan IIE (XC2S300E). (b) The Digilent Spartan 3 Starter Kit board, centered around a Xilinx Spartan III (XC3S400).

The class focuses on the design of a fairly complex hardware/software system, which the students implement on an FPGA board. This is consistent with the role of the course as a "capstone lab" in the Columbia Electrical Engineering and Computer Engineering programs. As I describe below, the first half of the class consists of cookbook-style lab assignments that teach the students the design tools. The second half of the class is devoted to the project.

As of September 2005, I have taught the class three times: twice at Columbia, and once at National Chaio Tung University (NCTU) in Hsinchu, Taiwan. At Columbia, the majority of students are fourth-year undergraduates completing an Electrical Engineering or Computer Engineering degree. The Taiwan group consisted of sixteen students, most of whom were Master's-level, and many of whom had worked or were working full-time in the electronics industry.

II. Hardware

Just as embedded hardware/software systems can take many forms, there are many possible vehicles for teaching a practical embedded systems class. At one point, I considered having the students only build simulations, a practical approach used in computer architecture courses, but I wanted the students to experience real hardware.

2004		2005	
Count in decimal on 7-segment LEDs	C	Count in decimal on 7-segment LEDs	C
Display "Hello world" using framebuffer	C	Terminal emulator using supplied video controller	C
TV typewriter	C	Reverse-engineer some VHDL	drawings
Count in hex on 7-segment LEDs	VHDL	Sum the contents of a memory	VHDL
Make framebuffer display characters	VHDL	Complex multiplier as OPB peripheral	VHDL
TV typewriter using character display	C & VHDL	SRAM controller for OPB (omitted at NCTU)	C & VHDL

Fig. 2. The six lab assignments.

One standard approach is to use a microprocessor development board. Frank Vahid has taken this approach with his course at University of California, Riverside [1], using the 8051. While this is a practical option, it tends to lead to software-centric thinking that does not consider hardware/software trade-offs.

Modern field-programmable gate arrays (FPGAs) offer many advantages for instruction, including flexibility, fast reprogrammability, and the capacity to implement large, fast digital designs. The two leading FPGA companies—Xilinx and Altera—offer comparable technology. Xilinx, however, appears to have the superior university program, so I decided to use Xilinx FPGAs for this course.

Many FPGA development boards are available, but most are designed for industrial use (and budgets) and can cost as much as US $5000 per board—beyond our price range.

At Columbia, my TA and I selected a board built by XESS Corporation: the XSB–300E (Figure 1a), which sells for about $900. Centered around a Xilinx Spartan IIE FPGA—the XC2S300E—with a raw capacity of roughly 300k gates, the board also has a wide variety of peripheral chips, including video input and output; Ethernet; USB 2.0; a serial port; SRAM, DRAM, and flash memory; and an audio CODEC. The peripherals were particularly attractive; using these peripherals would be a focus of the projects.

In Taiwan, we used a newer, smaller board made by Digilent (the Spartan 3 starter kit board, Figure 1b). This was satisfactory, especially given its $120 price, but greatly limited the range of projects as its peripherals are limited to an 8-color VGA port, a PS/2 keyboard interface, and an RS-232 port. This particular board comes in a few different configurations. The most common has an XC3S200 part, but this is too small to accommodate the Microblaze soft processor core, which every project has used. Instead, we paid a bit more and got boards with the larger XC3S400 part.

III. Textbooks

I have not found a satisfactory text for the class. In the first year, I used Vahid and Givargis [1], partially because Vahid had originally suggested the idea of the class to me and because the text embodies the philosophy of functions being implementable in either hardware or software. But the students and I found the book disappointing. It deliberately shies away from talking about any particular languages or platforms, making it useless as reference manual. The background material and ideas it contains are good, but the students do not find it terribly relevant to the project construction task.

Many texts are software-centric. Wolf [2], for example, discusses things such as the ARM instruction set and operating-system-level concepts such as processes. Simon [3] is even more focused on software, although a bit more practical than Wolf. Heath [4] is similar. Brown [5] has a more industrial focus and includes a large avionics example.

Many books are specific to a particular processor. Although practical, I find such an approach focuses too much on the idiosyncrasies of a particular instruction set. Books in this vein include Morton [6], which focuses on the Motorola 68HC11 series of microcontrollers. Lewis [7] targets the x86 architecture and ordinary PCs, which makes acquiring suitable hardware easy: most departments have a collection of old PCs that make suitable embedded targets. Barr [8] chooses an 80188-based board, although focuses mostly on C and C++. Pont's book [9] should have been titled *Embedded C on the 8051*. Peatman [10], by contrast, makes it clear that he focuses on the PIC18F452 processor. Incidentally, is is the only book I know of that comes with a (bare) PC board.

There is another family of texts that are more concept-oriented and targeted at graduate students. These are even more abstract that Vahid and Givargis and would probably frustrate my students. Examples include Gajski et al. [11], Jantsch [12], Marwedel [13], and the volume edited by Jerraya et al [14].

Noergaard's sprawling volume [15] tries to discuss just about everything, ranging from the difference between enhancement- and depletion-mode MOSFETs to HTTP methods. While a very interesting reference, it is too long to consider in its entirety in a single semester.

IV. Lab Assignments

As I mentioned above, I divide the class into two: during the first half, the students follow cookbook lab assignments meant to teach them how to use the design tools. During the second half, they design and implement projects of their own devising.

Figure 2 lists the six labs I have given over the last two years and the main language they need to use in each. The goal of these labs was to get the students familiar with using the tools through a sort of tutorial style. I provided detailed explanations of what to do as well as collections of files as a starting point. The results were mixed.

In the spring of 2004, I made the mistake of trying to balance software and hardware labs, making half of them

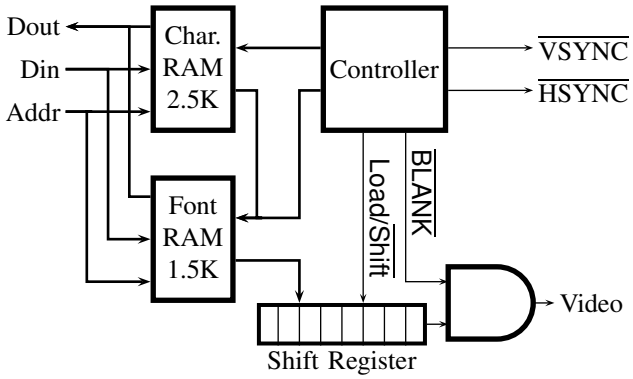


Fig. 3. A block diagram of a text-mode video controller. I describe the design of this peripheral in great detail to introduce the students to the design process.

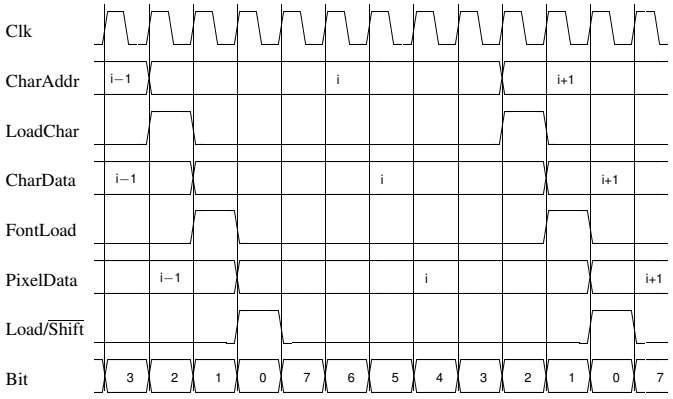


Fig. 4. A detailed timing diagram for the text-mode video controller. I teach the students such timing diagrams are crucial for designing functioning digital hardware.

software-dominated; the rest hardware. While this does reflect the class focus I had in mind, it was not well-matched to the students' backgrounds, which were heavily software-centric. I found myself teaching experienced programmers who were able to complete the first three labs with almost no effort at all. But digital design with VHDL stumped them: they did not have any real experience designing digital circuits, despite having taken a beginning digital design class. They were also flummoxed by the odd syntax of VHDL. Many of them resorted to trying to write VHDL as if it were C.

In the spring of 2005, I made the labs more hardware-centric. Again, the first two gave the students experience in low-level C programming and some experience with the tools, but the rest of the labs were mostly about design with VHDL.

Most students, when introduced to synthesizable VHDL, treat is as a programming language, but it is more a textual form of coding schematics and state machines. VHDL "statements" such as if-then-else and assignments are deceptive: they only provide a way of decomposing a function and do not behave like the imperative versions the students are familiar with. To try to help them avoid this error, I tried to emphasize a particular design methodology.

Two concepts distinguish hardware from software: structure and timing. While software has structure in the form of functions and classes, the structure in hardware is at a block-diagram level, reflecting its concurrent nature. Similarly, the software programming style is to ignore performance concerns until absolutely necessary and only concentrate on functionality, a technique that does not work in hardware. As a result, I teach the students a three-step hardware design process: draw a block diagram, such as the one for the text-mode video controller in Figure 3, draw a timing diagram (e.g., Figure 4), and then code it in VHDL. To get them started, the third lab in 2005 had them do this in reverse: we provided them with a clearly written VHDL description and asked them to draw the block diagram and a timing diagram for it.

The biggest challenge the students faced while doing the projects in 2004 was dealing with existing protocols such as the bus protocol spoken by the Microblaze soft processor (i.e., the OPB protocol) or the protocol spoken by the audio codec. To try to address this, the three hardware labs in 2005 were protocol-centric. The first was fairly easy: building a controller that would sum the contents of a small on-chip memory. Students had to understand the (very simple) interface to the memory, design a simple data path with controlling state machine, and understand how to use the VHDL simulator.

The second hardware lab involved interfacing with the OPB. I had students design and implement a simple memory-mapped peripheral that performed complex multiplication. We supplied the students with a combinational (one-cycle) 8×8 multiplier and asked them to construct a simple data path and controller that used the multiplier four times to compute the product of two complex numbers.

The final hardware lab was the most complicated, although still far from what the students would have to do while implementing their projects: an interface for an off-chip static RAM part. This is a typical problem: interfacing one protocol with another—in this case, the protocol of the OPB and the protocol of the static RAM. To keep things simple, I had them only map half of each 32-bit processor word to match the 16-bit width of the SRAM chip. This would allow them to store data in the memory, but not execute code from it since the processor needs all 32 bits.

These labs definitely worked better in the second year, but there remains room for improvement. While all groups managed to complete the labs successfully, many seemed to forget the lessons they taught when doing the project. I often found myself answering questions with "we did that in lab four," which was disappointing. Furthermore, it remains the case that the students need more practice at hardware design and debugging the mess they have created.

At Columbia, the course spans a normal fourteen-week semester; in Taiwan, it was condensed to a single month in which the class met daily. To accommodate the tight schedule, I omitted the sixth lab assignment and scaled down the scope of the projects.

V. The Project

In my experience, students prefer working on projects of their own devising rather than what I could supply. As an example, when I taught the compilers class at Columbia for the first time, I had the students implement the Tiger language from Appel's Modern Compilers book, and the students hated it. The next year, I had the students design and implement a language of their own devising—a much more difficult procedure, but the students clearly enjoyed it far more.

For the projects, I break the class up into groups of between two and six students. A size of three seems optimal—any smaller and the project becomes too simple, any larger and the group starts to lose its cohesion and ability to communicate.

Overall, about 80% of groups have completed the project, meaning they have something working at the end that closely resembles what they set out to do. The remaining 20% have difficulties with group dynamics (e.g., the members hate each other), technical difficulties (one group spent their time trying to communicate with the USB controller, to no avail), or are just incompetent. The good students at Columbia are excellent; the bad students are awful.

I ask the project groups for four deliverables: a two-paragraph project proposal, a project design document, a demonstration on "75% day," and the final demonstration and report. Such deadlines are absolutely necessary to keep the students moving as otherwise they would work on other classes' shorter deadlines. Even so, four deadlines seems like it may not be enough; I plan to add a "50% day" next year.

I expect a resonable project to incorporate both software (C) and custom hardware (VHDL) and interface with at least one of the on-board (but off-chip) peripherals on the XESS board. Interfacing with one of the peripherals is relatively straightforward, but using two or more is difficult because of the odd shared bus structure of the XSB–300E board, which connects all of the peripherals to a common set of pins on the FPGA. Thus, to communicate with multiple peripherals, the students must build a controller that behaves differently depending on the peripheral being accessed (each has very different timing requirements), yet does so through a common set of pins. Without question, this is one of the most awkward aspects of the XSB–300E board.

Students usually start by proposing overly ambitious projects (at least in the US—the Taiwanese students were much more realistic, but this may have been because many were professionals). My teaching assistants and I have had to curtail countless proposals that incorporate MPEG encoding, a complete TCP/IP implementation, or other systems that are orders of magnitude more difficult than beginning students could realistically implement in half a term.

Below, I describe the majority of the successful projects students have completed over the last two years. Broadly, they fall into four classes: video, audio, networking, and "other." The majority of projects focus on one of these areas, but some of the more ambitious and successful projects incorporate, say, both video and networking.

A. Video Projects

I am a fan of video-centric projects, having built some as an undergraduate. They have certain advantages: they are visually satisfying when they work; they can often be debugged by inspecting the displayed image; they have substantial, but not insurmountable, real-time requirements; and VGA-style video signals are a simple protocol whose central idea (a raster image) is fundamental. To support video development, each workstation in our lab has two flat-panel displays: one connected to a Linux workstation; the other connected to the XSB–300E board and its video DAC.

Video games Simple video games make for excellent projects. Students have implemented games inspired by *Pac-man*, *Scorched Earth*, and a 3D maze game. For the Pac-man-like maze game, students designed and implemented a custom video generator capable of drawing sprites over a character display, much like the original Namco arcade game. The game logic, implemented in C, was primitive, but I was more concerned with their implementation of the custom hardware and the hardware/software interface.

Scorched Earth is an artillery game in which players take turns lobbing shells at their opponents' tanks over a 2D terrain. The students implemented custom graphics hardware that superimposed sprites for the tanks and shells over a terrain generator (each column has a height that corresponds to the line at which the sky ends and the ground begins) and a character generator for displaying the current score, gun inclination, and so forth. Again, the game logic was simple but successful. This project was the star of 2005; most students wanted to play it.

One group implemented a 3D maze game. I had them create a custom video controller that contained two numbers for each X coordinate: one that corresponded to the line at which the sky begins and the wall starts; the other that holds the line at which the wall ends and the floor begins. The students used a primitive raycasting technique to determine these numbers: from the player's position, they sent out a ray that goes until it hits a wall. The distance the ray travels indicates the size of the wall at that column (more precisely, it is the reciprocal of the distance). The students found it challenging to do this calculation using fixed-point arithmetic (the Microblaze does not have a hardware floating-point unit), but were ultimately able to achieve nearly a 20 fps frame rate.

Since we used the simpler Digilent Spartan 3 Starter Kit Board (Figure 1b) at NCTU in Taiwan, the range of projects the students could build was greatly restricted. I suggested they build simple video games and most groups did.

Chess Rather than spend time on the algorithm for playing chess, this group built a two-player chess game that could display the chessboard, let a player select a piece to move, show where it could be moved, and move it. They even implemented such complicated rules as pawn promotion and castling. As with most of the NCTU projects, this group adapted the video display code I provided (based on Figures 3 and 4) to display the board and pieces.

The four other NCTU video game projects were Tetris, Sokoban, a scrolling maze game loosely patterned on the Namco game Rally-X, and a two-player snake game. Each group modified the text-mode video controller code I had provided, adding color and changing the size of the characters.

Video Effects Processor Modern FPGAs have enough processing power to perform limited real-time image processing. One group put this to use by building a framebuffer with the ability to distort its output. Rather than simply reading the contents of memory in sequence for adjacent pixels, they added the ability to change the starting point and memory stride for each line. Such a set-up was able to transform, say, a rectangular picture into a triangular one, and be able to modify this distortion on-the-fly. The group had originally intended to perform this distortion on real-time video (the XSB–300E has a Philips video decoder chip), but ran out of time and only displayed the static contents of memory.

Digital Picture Frame This project, which was unsuccessful in 2005, decodes JPEG images and displays them on the screen. The easiest way is to perform most of the computation in software and only use hardware for the framebuffer. The XSB board includes a Compact Flash interface (a parallel bus protocol), so in theory it would be possible to read and display files from a digital camera, but no group has been successful at being able to read from a CF card.

One group attempted to port the independent JPEG library onto the Microblaze in the process of performing this project, but ran into serious size and complexity problems. In the future, I will advise any group that undertakes this project to write their JPEG decoding code from scratch and not worry about making it support all JPEG variants. There is also an obvious opportunity for hardware acceleration (i.e., the inner loop of the DCT).

Another group, this one in Taiwan, also attempted the digital picture frame project, this time with greater success. First, they implemented a frame buffer that used external SRAM (there is only 32K of on-chip block RAM on the XC3S400 used on the Digilent board) and had to grapple with the usual problem of simultaneous access from both the video system and the processor. They took the simplest route and added a mode bit that would blank the display and enable to processor to access it. Next, they took a small JPEG library written by Pierre Guerrier[2] and made it compile on the Microblaze. Finally, they added a mechanism for copying data from the RS232 port into memory in preparation for decompressing and displaying it. Unfortunately, little of this worked completely at the end of the class. The JPEG library, for example, just barely fit in the 16K of on-chip memory they were using.

Video Input Projects These are quite a bit more challenging than video generation projects. First of all, the video DAC is a much simpler chip than the Philips SAA7114H video decoder, which has a 140-page manual. Second, it is much easier to generate a signal than to understand one, especially when it comes from the real-world.

[2]Their source was http://www.es.ele.tue.nl/~mininoc/c_prog/djpeg_orig/

One group implemented a stereo depth extractor using the video decoding capabilities of the XSB–300E board. They pointed a video camera at a mirror nearly parallel to the centerline of the camera to generate a split image from two slightly different vantage points. They then looked for the two brightest spots on the image and used the difference in their location to compute the 3D location of the spot. To test this, they placed the camera in a black cardboard box and shined a laser pointer at a movable target. Although clearly not at the cutting edge of computer vision, the group was able to get interesting results.

Robot with Vision Perhaps the most unique project to date, this incorporated the XSB–300E board as the controller for a mobile robot built using Lego Mindstorms. In the end able to follow a black line drawn on a white piece of paper, the most unique aspect of this project was the successful use of video as vision. The group mounted a small video camera to the Mindstorms robot and fed the input to the XSB–300E board. The board decoded the video, posterize it to one bit per pixel, divided the image into nine rectangular regions, and used the relative number of black pixels in each region to decide whether to turn left, right, or to go straight.

Software running on the Microblaze analyzed the heavily decimated video input signal and transmitted simple commands through a serial port to an IR tower and the robot itself, whose controller was running a very simple program that took simple direction commands.

B. Audio Projects

Like video projects, audio projects engage the senses and therefore share some of the thrills of success and easy debugging of video projects. Compared to video, however, audio is nearly three orders of magnitude slower and one-dimensional, making it much easier to manipulate and presenting many more opportunities for elaborate signal processing.

The audio CODEC on the XSB–300E (an AKM AK4565: 50 kHz, 20 bits/sample, stereo) has a synchronous serial interface with a fairly simple protocol, although its configuration protocol, which goes through a separate synchronous serial interface that is, unfortunately, connected to the low-order data bits on the XSB–300E peripheral bus, was difficult for most of the students.

MIDI Synthesizer One of the most successful projects of 2004, a MIDI synthesizer leads to a nice combination of hardware and software. While it would be possible to perform the sound synthesis in software, its real-time requirements are sufficiently demanding and its computational complexity makes it simple enough to do in hardware. This group implemented both the standard FM synthesis algorithm and the Karplus-Strong plucked instrument algorithm. Both sounded remarkably good.

MIDI is an asynchronous serial protocol like RS-232, but at an unusual bit rate and generally transmitted through a current-loop designed to be terminated with an optoisolator to avoid noise. The group built a simple MIDI-to-RS-232 level

converter and used the soft UART core supplied by Xilinx to receive the protocol.

The MIDI protocol consists mostly of note-on and note-off messages. While fairly simple, managing polyphony with a finite number of oscillators is much easier to do in software, which the group did. Thus, the MIDI protocol was decoded in software and the synthesis was done in hardware.

Sound Effects Synthesizers FPGAs now have more than enough processing power to perform fairly complex audio-band signal processing. At least two groups have taken advantage of this by implementing various sound effect generators. One, for example, was designed to implement various effects, such as phasing and distortion, that worked well with input from an electric guitar. The algorithms for such filters are fairly straightforward; the groups implemented them in hardware and placed their parameters under software control.

Audio Spectrum Analyzer I was a little surprised by the speed of the FPGA for this project: one group implemented a real-time 1024-point FFT running at audio speeds. The majority of the algorithm was in software; they only used hardware to accelerate complex multiplication. Coupled with a nifty graphic equalizer-like display, this was an impressive project.

Pitch Detection Detecting the fundamental pitch of an audio signal, such as a voice, is a fairly interesting problem with applications to singer training. Two groups have attempted projects along these lines, with varying results. One group did the obvious and performed an FFT on audio input samples, but found that the linear bin arrangement of the FFT made for rather imprecise measurement at low frequencies. Another group attempted to implement an algorithm based on autocorrelation and got so far as to built a prototype in Matlab, but did not complete the project because of group dynamics.

C. Networking Projects

The XSB–300E contains an NE2000-compatible Ethernet chip, and a number of projects have used it for network communication. By sheer numbers, students seem to prefer audio and video projects, but the successful network projects have been quite impressive.

Internet Camera This project, which combined video and networking components, spoke the most protocols of any project to date. The students used the Philips video decoder chip to sample real-time video, decimate its resolution and frame rate, packetize it, and send it as UDP packets over Ethernet. On the receiving end—a standard Apple laptop—a simple Java program of their own devising received the packets and displayed them. I joked that the project amounted to a very expensive wire, but it was actually one of the most technically challenging projects that exemplified good engineering: it made something quite complicated look effortless.

Internet Audio In 2005, two groups built projects that communicated audio over the Internet. One built an Internet radio broadcaster that took audio in through the CODEC, packetized it, and sent it out via RTP. They connected an iPod and an Ethernet cable to the board and were able to listen to the iPod through the speakers on the Linux-based workstation running a standard *mplayer* program. Like the Internet camera project, this one went to pains to speak many standard protocols and ultimately made something very complicated look simple.

A similar project took on an even more complex protocol: SIP. Designed for Internet telephony, SIP is a standard protocol for establishing Voice-over-IP telephone calls. I was amazed when this group was able to hook up the board to the campus Ethernet network and make a friend's VoIP phone ring. The final result was a little disappointing—the audio quality was poor, which I attributed to some sloppy programming somewhere—but overall the project was a success.

D. Unique Projects

All the projects I described above used the XSB–300E board and its FPGA at the center. By design, in this class I have not focused on the electrical and physical challenges of embedded system design, but a number of groups decided to tackle these problems, too. The result has been a largely successful group of unique projects.

Automotive Projects Columbia participates in the Society of Automotive Engineers' Formula SAE competition, in which student groups design, fabricate, and race small cars built around motorcycle engines. While largely targeted at mechanical and automotive engineers, two groups in my class have done projects related to this effort.

The first, in 2004, built a vehicle telemetry system that gathered data from various sensors on the car (e.g., tachometer, oil pressure) and sent it through a wireless link. The group purchased the wireless transmitters and receivers, but built a small processor system on the car centered around a PIC microcontroller. This was a particularly challenging project for this class because it involved a number of analog signal conditioning circuits. While ultimately mostly successful, the group did have the problem of relying on the car itself—which was often unable to start—to demonstrate their project.

The 2005 group built a digital dashboard and controller for an automatic shifter. They worked with a pair of mechanical engineers who designed the mount and levers for a large (25A) solenoid that moved the mechanical gear shift lever (the gears on this motorcycle engine could be selected by moving a lever—designed to be operated by foot—up and down). The group also built a dashboard with LEDs displaying engine RPMs, the current gear, and a suggestion about whether to shift up or down. As is typical of beginning electrical engineering students, their fabrication skills were lacking, resulting in a tangled mess of wires and cold solder joints, exactly the sort of problems using an exclusively FPGA-based approach avoids. Alas, this group, too, was stymied by the engine not starting when they tried to demonstrate it to me, although it had been working earlier.

Scrabble Timer When the class started, an inventor happened to approach me about building a prototype of a timer, much like a chess timer, for the game of Scrabble—he had aims of selling the design to a large board game manufacturer.

Question	2004	2005
Amount Learned (out of five)	3.72	4.04
Appropriateness of Workload	3.33	3.64
Overall Quality	3.74	3.89

Selected comments from 2005:

"Tough class but learned a great deal. Recommended."

"I'd like to see a lecture that goes into more detail about the way that the various files definitions and programs are used to create the hardware. We end up learning it in pieces but a more detailed overview would be useful since the tools are a key component of understanding this class."

"The lectures didn't seem to serve as much help for the assignments and project."

Fig. 5. Course evaluation results for the Columbia classes. Numbers are averages, with 0=poor and 5=excellent.

I figured it would make a good project for the embedded systems class and it did. A group worked closely with the inventor, who had a clear idea of the behavior he wanted (he came to me with a multi-page document describing how it should work) but did not have the skills to implement it. This arrangement worked perfectly—the students got the opportunity to work for a client and in return were given very precise requirements and a helpful client.

Like the automotive groups, this group built a system centered around a PIC microcontroller. Its interfaces were simple: a collection of buttons numerous enough to require multiplexing and 4 line $\times$ 40 character LCD display module. While a very simple design, the quality of the software and attention to detail (the group put their project in an attractive box and spend a lot of time thinking carefully about the arrangement of buttons) made this project a stand-out.

All three of the groups that used PIC microcontrollers got little instruction from me. I do not discuss microcontroller programming in class, but the students were able to glean the information they needed from tutorials and web references. The students who have worked on such independent projects, not surprisingly, have been among the best in the class.

VI. Evaluation

As with all Columbia classes, students were asked to fill out a standard course evaluation form. The results are summarized in Figure 5. While the results are positive overall, and the ratings improved uniformly in the second year, the main negative complaints were that the course required a lot of work (which is certainly true) and that my lectures were not that relevant to the labs and project. Specifically, they wanted much more instruction on how to use the CAD tools, which are very complicated. I had attempted to address this issue through detailed lab writeups, which described the operation of the tools in detail, but some students obviously prefer to be shown something.

VII. Conclusions

The embedded systems class I have described remains a work-in-progress, but has been fairly successful. The word-of-mouth among students has been excellent, to the point where it has clearly siphoned off students from other, competing classes. An informal poll suggests students prefer the mix of hardware and software and the ability to choose their projects.

I have put all the class materials on the web, including all slides, handouts, lab assignments, lab template files, and students' project files and reports. All can be found at http://www.cs.columbia.edu/~sedwards/classes.html.

I have not been able to address to my satisfaction the balance between practical knowledge and fundamental understanding. There is a plethora of practical knowledge the students need to be able to execute their projects, ranging from VHDL coding styles to how to get the Xilinx tools to work, but I feel the class is overly demanding of knowledge of such trivia. Frustratingly, the students actively complain that I spend too little time lecturing about such details. While this is very realistic and class is meant to be a capstone class in which students bring together the knowledge they have gained during their careers as undergraduates, it still frustrates me that I feel they are learning trivia that will be out-of-date in a year.

The more fundamental ideas I see in practical embedded system design—the balance between top-down and bottom-up design necessary to build high-performance systems, the ability to debug, the ability to seek and find the information you need, and the ability to understand and reverse-engineer poorly-written documentation—are subtle, difficult to convey, and not the sort of thing you can easily ask on a test. My hope is that these more subtle ideas will enter in the students' thinking during the process of implementing these complex projects, because it is unlikely they will learn them from a lecture or a book.

References

[1] F. Vahid and T. G. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*. New York: John Wiley & Sons, 2001.

[2] W. Wolf, *Computers as Components: Principles of Embedded Computer Systems Design*. San Francisco, California: Morgan Kaufmann, 2000.

[3] D. E. Simon, *An Embedded Software Primer*. Reading, Massachusetts: Addison-Wesley, 1999.

[4] S. Heath, *Embedded Systems Design*. Oxford: Newnes, 1997.

[5] J. F. Brown, *Embedded Systems Programming in C and Assembly*. New York, New York: Van Nostrand Reinhold, 1994.

[6] T. D. Morton, *Embedded Microcontrollers*. Prentice Hall, 2001.

[7] D. W. Lewis, *Fundamentals of Embedded Software*. Prentice Hall, 2002.

[8] M. Barr, *Programming Embedded Systems in C and C++*. Sebastopol, California: O'Reilly & Associates, Inc., 1999.

[9] M. J. Pont, *Embedded C*. Addison-Wesley, 2002.

[10] J. B. Peatman, *Embedded Design with the PIC18F452 Microcontroller*. Prentice Hall, 2003.

[11] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[12] A. Jantsch, *Modeling Embedded Systems and SOC's*. Morgan Kaufmann, 2004.

[13] P. Marwedel, *Embedded System Design*. Kluwer, 2003.

[14] A. A. Jerraya, S. Yoo, D. Verkest, and N. Wehn, Eds., *Embedded Software for SoC*. Kluwer, 2003.

[15] T. Noergaard, *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes (Elsevier), 2005.